HETED-HÉT

commodore 64

1·2·3

ELŐSZÓ A MÁSODIK KIADÁSHOZ

Mindannyian, akik részt vettünk a Hetedhét Commodore 64 című könyv létrehozásában, nagy örömmel vettük tudomásul, hogy munkánkat a közönség is, a kritika is kedvező fogadtatásban részesítette. Olyannyira, hogy hamar elő kellett rukkolnunk a második kiadással, mert az előző alig pár hét alatt mind egy szálig elfogyott.

Olvasóink kérésére - és a nyomdai átfutási idő csökkentése érdekében - az eddigi három füzetet egy kötetben foglaltuk össze, s ahol észrevettük, javítottuk az előző kiadás hibáit is. Az új kiadás árát így alacsonyabban szabhattuk meg, mint az előzőé volt. Reméljük, olvasóink emiatt nem vonják meg rokonszenvüket a könyvtől...

Budapest, 1985. szeptember

Novotrade RT.

HETEDHÉT

Commodore 64

HARMADIK HÉT

EZ AZ UTOLSÓ KÖTET

Egy kötettel ismét beljebb jutottunk a Commodore 64 rejtelmeibe. Természetesen messze vagyunk még a céltól - már ha egyetértünk abban, hogy célunk a gép megismerése. Hadd oszlassunk el egy talán felmerülő tévhitet: ennek a harmadik kötetnek a végén sem lesz belőlünk tanársegéd a BASIC nyelv egyetemi tanszékén. Arra azonban alighanem jó ez a könyv, hogy a kíváncsiságunkat fölkeltse és ébren tartsa. A számítógép pedig majd elvégzi a többit.



1. NAP — ASC ÉS CHR$, A KÉT IKERTESTVÉR

Tudjuk már régóta, hogy a Commodore 64-en a színeknek kódszámuk van. Azonban nemcsak ezeknek, hanem az összes betűnek, számjegynek, jelnek is.

Az A betű kódszáma 65, az & jele 38; a teljes listát a könyv végén találjuk meg.

Ahogy a telefonnál, itt is kétféle "tudakozó" van: ha a kódszámot ismeri az ember, és a karakterre kíváncsi, akkor a következőt kell begépelnie:

```
? CHR$(N)
```

ahol az N helyére a kódszamot kell beírni. Tehát:

```
? CHR$(191)
```

mire a RETURNre a gép a ▚ karakter kiírásával válaszol.

Ha pedig visszafelé kérdezünk, tehát a kódszámra vagyunk kíváncsiak, akkor a CHR$(N) helyett az ASC("N") kifejezést kell használni. Ha beírjuk a következő sort:

```
? ASC("K")
```

akkor a RETURN megnyomasa után a gép a 75-ös számot írja ki - ez a K betű kódszáma. Ugyanígy 75-öt ír ki akkor is, ha azt mondjuk neki: ? ASC("KALAP"), mivel ez az utasítás csak a stringek első karakterét vizsgálja, tovább nem. Épp erre épül egy játékunk, amelyet kicsit később írunk le.

A CHR$ táblázat nemcsak a karakterek kódszámát tartalmazza, hanem a többi billentyűfunkcióét is, tehát a színek megváltoztatásának, a cursor-mozgatásnak a számát, stb. stb.

Érdekes kis képernyő-tüzijátékot rendezhetünk, ha az összes CHR$-funkciót felvonultató programocskát lefuttatjuk:

```
10 POKE 53281,0
20 FOR N=0 TO 255
30 PRINT N,CHR$(N),
40 FOR J=1 TO 200:NEXT J
50 NEXT N
```

Jókora összevisszaság is keletkezik olykor, de néha szabályos táblázattá alakul a dolog - attól függően, hogy az illető CHR$-nek éppen mi a funkciója.

Ha lefutott a program, listáztassuk ki, és utána írjuk be:

```
? CHR$(14)
```

és RETURN.

Mi történt? A program betűi mind kisbetűvé változtak, mintha együttesen lenyomtuk volna a C= és a SHIFT gombot. Úgy is van: a CHR$(14)-nek pontosan ez a feladata. Ha tehát azt akarjuk, hogy egy programunkban a képernyőfeliratok kisbetűvel jelenjenek meg, valahová az elejére be kell írnunk: ? CHR$(14). Ha pedig ismét nagybetűre akarunk váltani, akkor a CHR$(142)-t kell használnunk a programban.

Mit kell tennünk, ha kódszámot keresünk arra, hogy a cursor színe bíborra változzék (és nincs kéznél a táblázat sem)? Meg kell kérdezni a géptől, mert neki pontos leltára van. Írjuk be:

```
? ASC("{CTRL+5}")
```

és RETURN. A gép válasza:156.

Ezután gépeljük be: ? CHR$(156), és a READY már bíbor színnel jelenik meg a képernyőn. Ha tehát egy szöveget bíborral akarunk kiíratni, akkor egy újabb lehetőségünk van (az előző az volt, hogy magába a stringbe írtuk be azt a karaktert, amely a színváltást előidézte). Bárhol a programunkban használhatjuk e kódszámokat, a számitógép engedelmesen végrehajtja ezt a kódolt utasítást.

El lehet játszani ezzel is: ha az ember a táblázatot jól elteszi szem elől, és mindenféle ? ASC és ? CHR$ kódokat gépel be, és megpróbálja kitalálni, hogy mi fog történni, ha megnyomja a RETURNt.

Mi történik például, ha azt írjuk be: ? CHR$(147) - és RETURN? Mindjárt kiderül, mert a következő programunknak ez az első sora.

```
0 REM ASC-CHR$
10 PRINT CHR$(147)
20 POKE 53281,1
30 FOR J=149 TO 156
40 PRINT CHR$(J)
50 FOR I=65 TO 90
60 PRINT TAB(6)CHR$(I);
70 NEXT I
80 NEXT J
90 GOTO 30
```

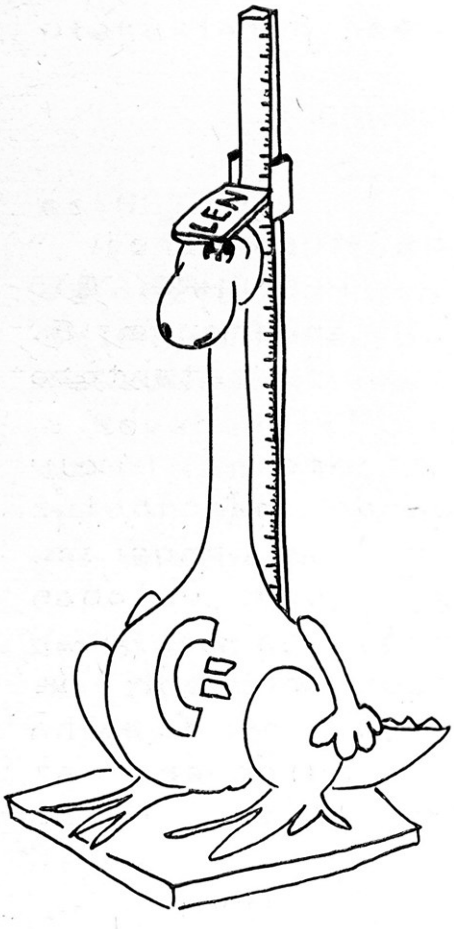
Az ASC és a CHR$ olyanok, mint egymás tükörképei - pontosan megfelelnek egymásnak. ? ASC(CHR$(70)) eredménye 70, aminthogy ? CHR$(ASC("F")) eredménye F. Az F és a 70 a számítógép számára szorosan összetartozó két adat.

Nos, akkor rátérhetünk az ígért játékra, amely azt vizsgálja, hogy hány B betűvel kezdődő hárombetűs szót ismerünk. (Ez egy nagyon egyszerű kis program, könnyen becsapható: csak azt vizsgálja, hogy valóban B-vel kezdődik-e az INPUTban kapott szó, és valóban három betűből áll-e. Tehát ugyanazt a szót többször is elfogadja, és összefüggéstelen betűcsoportokat is, ha B-vel kezdődnek. Hát persze, hiszen különben az Értelmező Szótár B betűs szócikkét kellett volna kijegyzetelnünk és betáplálnunk...)

```
0 REM HAROMBETUS, B-VEL KEZDODO SZAVAK
10 PRINT CHR$(147)
20 PRINT"IRJ BE HAROMBETUS SZAVAKAT B-VEL!"
30 PRINT "HA NEM TUDSZ TOBBET, IRJ 0-T!"
40 LET I=0
50 INPUT "NOS";SZO$
60 IF ASC(SZO$)=48 THEN GOTO 120
70 IFASC(SZO$)<>66 THEN GOTO 100
80 IF LEN(SZO$)<>3 THEN GOTO 100
```

```
90 LET I=I+1:GOTO 50
100 PRINT "EZ NEM JO!"
110 GOTO 50
120 PRINT I;"SZOT SOROLTAL FEL."
130 END
```

Itt is egy csomó dolog ismerős és világos, egy dolog uj csupán: a 80-as sorban a LEN. Ez nem az ismert rostnövény, a textilipari alapanyag, hanem megint csak egy BASIC kifejezés, a "length", azaz "hosszúság" angol szó rövidítése. Ez a BASIC utasítás megszámolja a stringekben található karaktereket, és kérésre ki is írja. Íme:




```
10 INPUT A$
20 PRINT LEN(A$)
```

és kész. Ez természetesen nem lehet korlátlan hosszúságú, a maximális hossza legfeljebb 2 képernyősor, azaz 80 karakter - de hát a stringjeink is csak ilyen hosszúak lehetnek, tehát ez nem okoz nehézséget.

A fenti program természetesen átírható úgy, hogy ne 3, hanem mondjuk 9 betűs szavakat kérdezzen, és ne B betűvel kezdődjenek a szavak, hanem Q-val - ez a gépnek nem okoz nehézséget, legföljebb a játékosnak... A nehézséget így tetszés szerint lehet módosítani, tehát egészen jó kis társasjáték lehet belőle. Csináljuk is meg - legyen ez a mai feladat.

Néhány nappal korábban már megpendítettük a vektorok és mátrixok témáját, most lássunk egy olyan programot, amely leltárt csinál egy osztályban található ceruzákról.

Az osztály, ha mindenki a helyén ül, olyan, mint egy mátrix: három padsor, öt-öt paddal, mindegyikben két-két gyerek:

```
*  *    *  *    *  *
*  *    *  *    *  *
*  *    *  *    *  *
*  *    *  *    *  *
*  *    *  *    *  *
```

és némelyiknek több ceruzája is van, másoknak meg egy sincs.

Ha a valóságos helyzetet felírjuk, egy tetszetős mátrixot kapunk, sorokba, oszlopokba rendezve.

	OSZLOPOK					
SOROK	0	1	2	3	4	5
0	3	0	0	2	0	2
1	2	1	0	2	1	4
2	0	0	1	2	0	0
3	0	1	1	0	3	0
4	2	1	3	1	1	1

Nevezzük el a mátrixunkat így: CERUZA. Ha dimenzionálni akarjuk, így határoljuk be: DIM CERUZA(4,5), hiszen 5 sorunk és 6 oszlopunk van, és az elsőnek 0 a száma.

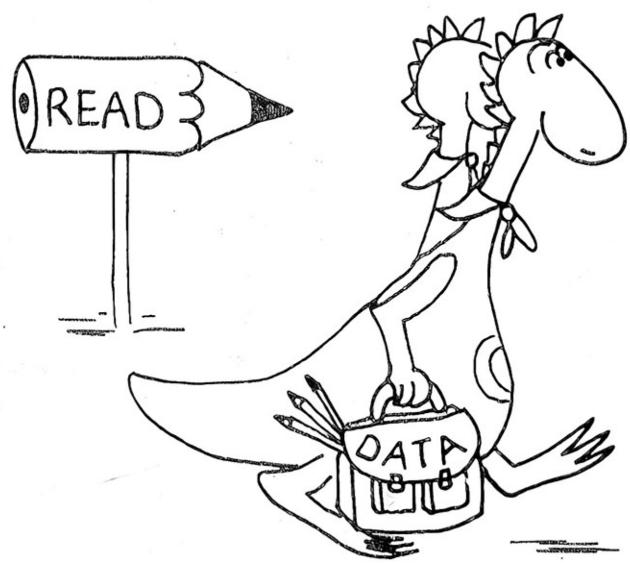
Az itt következő program ezekből az adatokból az ismert módon próbálja kiszámítani a ceruzák számát. Nézzük meg figyelmesen!

```
5 DIM CERUZA(4,5)
10 OSSZ=0
120 FOR Y=0 TO 5
130 FOR X=0 TO 4
140 OSSZ=OSSZ+CERUZA(X,Y)
150 NEXT X
160 NEXT Y
170 PRINT OSSZ
```

Ez a program összeadja a mátrix elemeinek értékét, a program végén az OSSZ változó megmondja, hogy az osztályban összesen hány ceruza van.

Ha tudja. De hát honnan is tudná: hiszen nem mondtuk meg neki. Ha így lefuttatjuk, kiír egy nullát - a maga részéről ez minden, amit tud az ügyről. Nulla darab ceruzáról informáltuk, ne kívánjunk tőle többet.

Aki eléggé szemfüles, az észrevehette, hogy ennek a programnak a közepén egy jókora hézagot hagytunk a sorszámozásban: azért, hogy elhelyezhessük oda az adatbeolvasási programrészt. Az adatokat eddig mindig egyesével adogattuk be a programozás közben, vagy az INPUTokban; most meglátjuk, hogyan lehet egy csokorban beadni a gépnek az összes adatot.



Adatok angolul: DATA - ezt nem nehéz magyar anyanyelvűnek sem megtanulni.

A programba külön sorban beírjuk a megfelelő adatokat (a sor elején csak ennyi áll: DATA), és a számítógépet egy alkalmas utasítással rábírjuk, hogy olvassa be ezeket. Az alkalmas utasítás ez: READ (ejtsd: ríd, magyarul: olvass!)

Az adatbeolvasási résszel és az adatokkal együtt a programunk végső formája ez lesz:

```
1 REM MATRIX-DATA
5 DIM CERUZA(4,5)
10 OSSZ=0
20 FOR Y=0 TO 5
30 FOR X=0 TO 4
40 READ CERUZA(X,Y)
50 NEXT X
60 NEXT Y
70 FOR X=0 TO 4
80 FOR Y=0 TO 4
90 PRINT TAB(6) CERUZA(X,Y);
100 NEXT Y:PRINT CERUZA(X,5)
110 NEXT X
120 FOR Y=0 TO 5
130 FOR X=0 TO 4
140 OSSZ=OSSZ+CERUZA(X,Y)
150 NEXT X
160 NEXT Y
170 PRINT"AZ OSZTALYBAN OSSZESEN"OSSZ"CERUZA VAN."
200 DATA 3,2,0,0,2,0,1,0,1,1,0,0,1,1,3
210 DATA 2,2,2,0,1,0,1,0,3,1,2,4,0,0,1
```

Látnivaló, hogy két FOR...NEXT ciklus ágyazódik egymásba; az előbbi, csonka programban kettő, itt háromszor kettő.

Az új ciklusok (a 20-astól a 60-as sorig) beolvassák a DATA sorok adatait, majd ki is írják a képernyőre (erről gondoskodik a 70-110 sor közötti programrész).

A READ utasítás megkeresi a programban a DATA kezdetű sorokat, és sorban, ahogy jönnek, behelyettesíti az adatokat a megfelelő helyekre. (Vigyázni kell a DATA sorok írásánál: az egyes adatok közé vesszőt kell tenni, a sorok végére viszont nem szabad! Egy DATA-sor, mint bármely más programsor a Commodore 64-en, 80 karakter hosszú lehet.)

A READ nem lehet meg DATA nélkül. Erre nézve bizonyára mindenki szerzett már egy bizonyos tapasztalatot. Elég egy vigyázatlan mozdulat, amikor a cursor a képernyőn egy sorban áll a READY üzenettel, amelyet a gép adott. Ha ilyenkor véletlenül megnyomjuk a RETURNt, a gép egy eddig megmagyarázhatatlannak tűnő üzenetet adott: OUT OF DATA ERROR. (Magyarul: "nincs DATA" hiba.) De hát miféle DATAról van szó? A

számítógép a READY szót így értelmezte: READ Y, vagyis olvasd be az Y-ra vonatkozó adatokat. Ilyet pedig nem talált, ezért felelt hibaüzenettel...

Ez később is így lesz, ha véletlenül kevesebb DATA áll rendelkezésre, mint amennyit keres a program. Vigyázzunk hát, amikor DATA-sorokat írunk, nehogy adósai maradjunk a gépnek.

A programban egyébként többször is szerepelhet READ utasítás, és ha ezekhez kevés adat tartozik, azokat összegyűjthetjük egyetlen DATA sorba, csak az a lényeg, hogy az adatok sorrendjére vigyázzunk.

No hát, akkor olvassuk a DATAkat.

```
10 READ K%
20 DATA 19
30 PRINT K%
```

Itt is ügyelnünk kell a változókra: a valós, az egész és a szöveges elkülönítendő, különben SYNTAX ERRORt kapunk a géptől.

Mindezek ismeretében olvassuk át még egyszer a ceruzák nyilvántartásáról szóló programot, és játsszunk el azzal, hogy a DATA sorokban kicseréljük a számokat, majd megpróbáljuk előre kitalálni, hogy a mátrixban hol fog jelentkezni az új érték!

HA holnap nem kell korán kelni, ÉS este a család nem akar tévét nézni, AKKOR 8 óra után is lehet gyakorolni a számítógépen.

Kecsegtető kilátás; ráadásul olyan formában, hogy BASIC programban is megfogalmazható.

No persze, az IF...THEN szerkezet - de hát ott egy feltételről volt szó, itt meg kettő van!

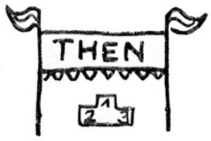
Sebaj: az ÉSnek angolul is van megfelelője, és a BASICben is: az, hogy AND. A mondat, amelyet az első bekezdésben írtunk, így írható fel magyar-BASIC öszvérnyelven:

IF holnap nem kell korán kelni,
AND a család nem akar tévét nézni,
THEN 8 óra után is lehet gyakorolni a számítógépen.

Ez persze ebben a formában még nem foglalható programba, viszont jól szemlélteti, hogy a BASIC program egyszerre több feltételt is képes vizsgálni. Az AND kifejezés azt jelenti, hogy a program csak akkor tér át az "igen" ágra, ha mindkét feltétel teljesül: ha tehát sem korán kelni nem kell holnap, sem a család nem akar tévét nézni. Az "igen" ág itt két "nem" szót tartalmazó feltételből áll, de ugye ez nem téveszt meg senkit?

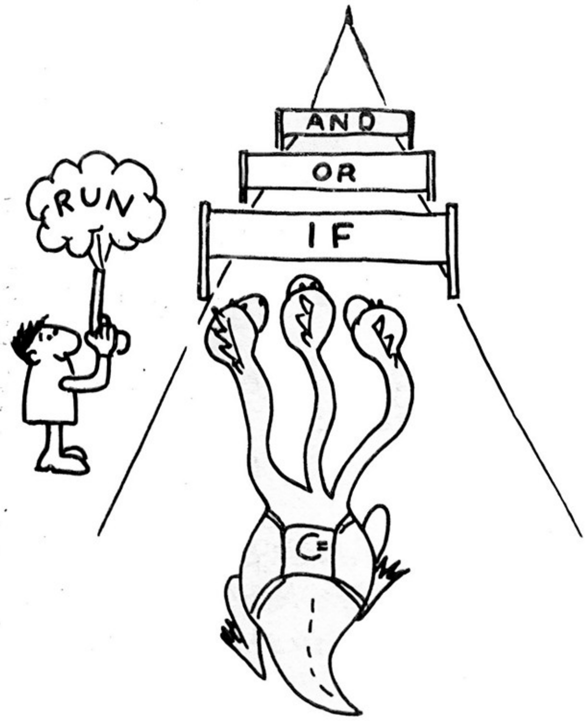
Az AND azonban még mindig nem az utolsó a számítógépes feltételes mód szavai közül: ott van még az OR is, amely azt jelenti: VAGY (nem ige, hanem kötőszó!). Vagyis elő lehet írni a számítógépnek azt is, hogy elég, ha a két - vagy több - feltétel közül az egyik teljesül!

Először nézzük a magyar nyelvű példát, mert azért ez logikai feladat, mi meg inkább számítógépes szakemberek vagyunk.



IF holnap szombat lesz,
OR holnap vasárnap lesz,
THEN nem kell korán kelni.

Ha tehát bármelyik feltétel teljesül a kettő közül, akkor megnyílik az "igen" ághoz vezető kapu - azaz a THEN utáni utasítások lépnek hatályba.



IF esik az eső,
OR erős szél fúj,
THEN nem megyünk kirándulni.

Melyik itt az "igen" ág? Az, ha elmegyünk kirándulni, vagy az, ha nem? Az "igen" - mint tudjuk - mindig az az ág, amelyik a THEN után következik, tehát ebben a példában a "nem" ágon megyünk kirándulni, és az "igen" ágon nem. Ugye világos?
Akkor menjünk tovább. Ugyanis az ORt és az ANDet össze is lehet kapcsolni..

IF holnap szombat lesz,
OR holnap vasárnap lesz,
AND a család nem akarja nézni a tévét,
THEN este 8 után is lehet gyakorolni a számítógépen.

Itt a három feltétel közül kettőnek kell teljesülnie: egyrészt annak, hogy holnap szombat vagy vasárnap legyen, másrészt, hogy ne akarjanak tévézni.

Ebben a példában a vizsgálat úgy folyik, hogy először megnézzük: milyen nap lesz holnap, és ha sem szombat, sem vasárnap nem lesz, akkor a "nem" ágra kanyarodunk; azaz este nyolc után nem komputerezünk. Ebben az esetben mindegy, hogy a család akar-e tévét nézni, vagy nem.

Vigyáznunk kell persze, hogy a feltételeket a megfelelő sorrendben fogalmazzuk meg, mert számos

választási lehetőség nyílik, és könnyen mellékvágányra futhatunk.

IF a család nem akarja nézni a tévét,
AND holnap szombat lesz,
OR holnap vasárnap lesz,
THEN este 8 után is lehet gyakorolni a számítógépen.

Itt első pillantásra semmi különbség nincs,csak a sorrend változott. Pedig ebben az esetben a család tévénézési szándékát csak az első estére vizsgáljuk, a másodikra nem! Igy téves következtetésre juthatunk, aminek az lesz a vége, hogy kidobnak a tévékészülék elől.

Ebben a példában a félreértést úgy lehet eloszlatni, hogy az együtt vizsgálandó feltételeket zárójelbe foglaljuk: IF a család nem akarja nézni a tévét, AND (holnap szombat lesz, OR holnap vasárnap lesz), THEN satöbbi. Ilyenkor a zárójel hatására a vizsgálat kiterjed a másik feltételre is.

Gondoljuk végig ezt a példát, és utána is gondoljuk végig ilyen szempontból minden egyes programunkban az IF... OR... AND... THEN szerkezeteket, mert előfordulhat, hogy ezen múlik egy programunk sikere vagy kudarca.

De elég ezekből a földhözragadt példákból, térjünk rá a BASIC nyelvre, és nézzük meg ott a feltételes módot.

```
IF A<7 AND Q=19 THEN GOTO 100
```

Ez a programsor csak akkor küldi a programot a 100-as sorra, ha A értéke kisebb 7-nél, Q pedig 19-cel egyenlő. Ha bármelyik feltétel nem teljesül, a program a következő sorra fut rá, nem a 100-ra.

```
IF B0$="K" OR B0$="L" THEN END
```

Itt a B0$ lehet K is meg L is, a program mindkét esetben befejeződik.

```
IF (A=19 AND B0$="K") OR B0$="L" THEN GOTO 100
```

Ekkor a program két esetben ugrik a 100-as sorra: ha a zárójelben foglalt feltételek együtt teljesülnek, vagy ha a zárójelen kívüli feltétel teljesül. Ekkor tehát elég, ha B0$="L".

```
IF A=19 AND (B0$="K" OR B0$="L") THEN GOTO 100
```

Itt egyrészt A-nak 19-cel kell egyenlőnek lennie, másrészt a zárójelben foglalt feltételek valamelyikének kell teljesülnie. Tehát a program akkor ugrik a 100-as sorra, ha A=19 és B0$="K" vagy ha A=19 és B0$="L".

No most, előadódhat olyan eset is, hogy csak akkor akarjuk a programot a 100-as sorra ugratni, ha A nem egyenlő 19-cel, minden más esetben igen. Ez például így fogalmazható meg:

```
200 IF A=19 THEN GOTO 220
210 GOTO 100
220 PRINT "A ERTEKE =", A
```

Ezt persze egyszerűbben is megírhatjuk:

```
200 IF A <> 19 THEN GOTO 100
210 PRINT "A ERTEKE =",A
```

A feltételekre vonatkozó logikai kifejezések sorába tartozik még a NOT (azaz: "nem") utasítás is, amely lényegében ugyanazt jelenti, mint a beszélt nyelvben. IF NOT - ez annyit jelent: "ha nem igaz, hogy..." és utána következhetnek a feltételek. A legegyszerűbb példa: NOT (A=19) - ez ugyanaz, mintha azt mondanánk: A<>19.

Ugyanaz, de mégsem egészen, hiszen pár karakterrel többet kellett gépelnünk hozzá. Tehát ez a NOT csak arra jó, hogy megnehezítse a dolgunkat? Korántsem.

```
IF NOT (A=B OR A<B) THEN GOTO 100
```

Ez esetben csak akkor ugrunk a 10-es sorra, ha A>B. A következő példában mikor ugrik a program 100-ra?

```
IF NOT (C<A AND D>A) THEN GOTO100
```

Gépeljük be az alábbi programot!

```
0 REM MIND ERTELMES?
10 POKE 53280,5:POKE 53281,13
20 PRINT"█":REM CLR/HOME,CTRL+1
30 A$(1)="EN"
40 A$(2)="EZT"
50 A$(3)="MONDTAM"
60 A$(4)="MAR"
70 A$(5)="EGYSZER"
80 A$(6)="TALAN"
90 A$(7)="NEKED"
100 PRINT:PRINT"  TUDOK 7 OLYAN SZOT,"
110 PRINT"  AMELYEKBOL, HA MONDATOKAT"
120 PRINT"  KESZITESZ, MINDEN MONDATOD"
130 PRINT"  ERTELMES LESZ."
140 PRINT"  HA NEM HISZED, JARJ UTANA!"
150 PRINT:PRINT
160 PRINT"IRD IDE A SZAMOKAT 1-TOL 7-IG"
170 PRINT"  TETSZOLEGES SORRENDBEN!"
180 PRINT:PRINT
190 FOR I=1 TO 7:INPUT B(I):NEXT I
200 PRINT:PRINT
210 FOR I=1 TO 7
220 PRINT" ";A$(B(I));
230 NEXT I:PRINT"."
240 GOTO 150
265 PRINT:PRINT
```

Ebben a programban minden elem ismerős, az utolsó vesszőig; legföljebb a 220-as sorban tűnhet egy kicsit gyanusnak e kifejezés: A$(B(I)). Ez, ha nem is teljesen

idegen gondolat, mégis szokatlan lehet az első pillantásra. Gondoljuk tehát végig.

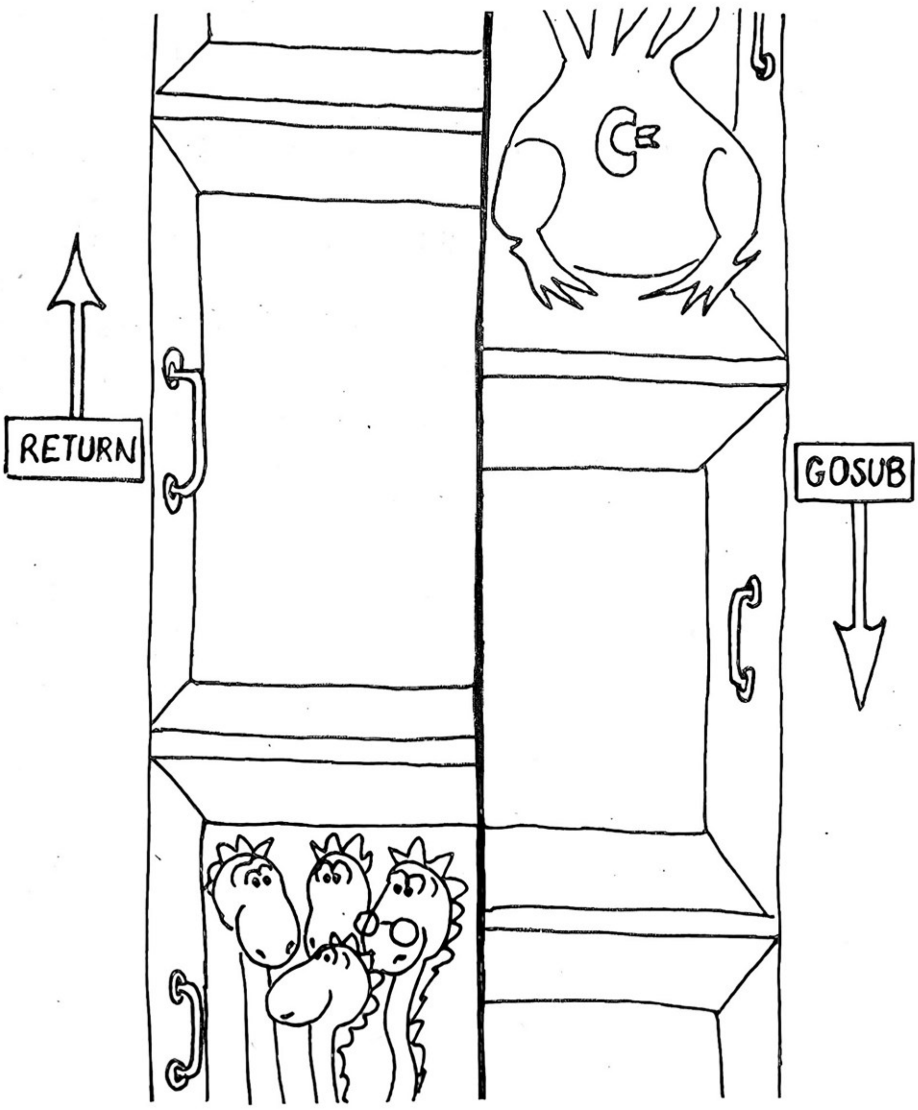
Itt az A$ indexét egy másik vektorelemmel határoztuk meg. Ez teljesen szabályos; a változók indexében (azaz a zárójelen belül) bármilyen kifejezés állhat; az a lényeg, hogy az értéke egész szám legyen, és ne essen kívül a tömbünk határain. A B(I) esetében ez a két feltétel teljesül is.

Ha eljátszottunk a programmal, és elámultunk azon, hogy milyen határtalanul rugalmas a magyar szórend (hiszen ezt a hét szót összesen 1*2*3*4*5*6*7=5040-féleképpen lehet más és más sorrendbe rakni), szóval ha már eléggé ismerjük a program működését, írjuk át a következőképpen!

```
0 REM MIND ERTELMES?
10 POKE 53280,5:POKE 53281,13
20 PRINT"■"
30 A$(1)="EN"
40 A$(2)="EZT"
50 A$(3)="MONDTAM"
60 A$(4)="MAR"
70 A$(5)="EGYSZER"
80 A$(6)="TALAN"
90 A$(7)="NEKED"
100 GOSUB 160
110 GOSUB 230
120 PRINT:FOR I=1 TO 7
130 PRINT" "A$(B(I));
140 NEXT I:PRINT"."
150 GOTO 110
160 REM KIHIVAS
170 PRINT:PRINT"  TUDOK 7 OLYAN SZOT,"
180 PRINT"  AMELYEKBOL, HA MONDATOKAT"
190 PRINT"  KESZITESZ, MINDEN MONDATOD"
200 PRINT"  ERTELMES LESZ."
210 PRINT"  HA NEM HISZED, JARJ UTANA!"
220 RETURN
230 REM VALASZTOTT SZAMSOR
240 PRINT:PRINT
250 PRINT"IRD IDE A SZAMOKAT 1-TOL 7-IG"
260 PRINT"  TETSZOLEGES SORRENDBEN!"
270 FOR I=1 TO 7:INPUT B(I):NEXT I
280 RETURN
```

Itt már van újdonság: a 100-as és a 110-es sorban a GOSUB. (Az angol kiejtése goszab, de sokan magyarosan gosubnak mondják.) Ez, akárcsak a GOTO, két szóból állt össze, a GO ("menj") és a SUBroutine ("al-rutin") szavakból. A rutin jelentését csak röviden mondjuk el, mert igazából a gyakorlatban derül ki: olyan programrész, amely többször is sorra kerül a programban, vagy esetleg csekély változtatásokkal más programba is áttehető. A szubrutin ennek az a változata, amely GOSUB utasítással hívható meg. Lényeges tulajdonsága, hogy a végén egy RETURN (ejtsd: ritörn; jelentése: térj vissza!) utasítás áll - ez nem tévesztendő össze a RETURN billentyűvel!

A GOSUB tulajdonképpen hasonlít a GOTO-hoz, egy lényeges különbséggel: a szubrutin végén a program futása ott folytatódik, ahonnan a GOSUB hatására elugrott; erre ad parancsot a RETURN. Ha tehát a 100-as sorban ez áll: GOSUB 2000:PRINT A$(1), akkor a program a 100-as sorra érve leugrik a 2000-esre, végrehajtja az ott kezdődő szubrutint, aztán visszaugrik, és kiírja az A$(1) értékét, akármi legyen is az.



Ha ugyanabban a programban többször is alkalmazzuk a GOSUB utasítást, ez vonatkozhat mindannyiszor ugyanarra a szubrutinra is, meg különbözőkre is; a RETURN mindig oda fordítja vissza a programot, ahonnan legutóbb elugrott.

A GOSUB hallatlanul jól használható a BASIC programokban; tiszta csoda, hogy eddig meg tudtunk lenni nélküle. A következő napon már olyan programot is fogunk vizsgálni, amely egy szubrutint többször használ fel. Egyúttal pedig választ kapunk egy régóta esedékes kérdésre is.

Ha az ember ránéz a Commodore 64 mikroszámítógépre, azonnal szemébe tűnik, hogy a billentyűzet két részre oszlik. A nagyobbik területen a szorosan egymás mellett elhelyezett, barna billentyűkön az ábécét, a számokat, jeleket és egyebeket látjuk; ezeket szinte kivétel nélkül használtuk is már.

Ott van azonban egy sokkal kisebb billentyű-csoport is a jobb szélen, jól elkülönítve, a színük is más, a méretük is, a jelölésük is: f1/f2, f3/f4, f5/f6, f7/f8 olvasható rajtuk.

Ezeket a billentyűket eddig úgy kerültük, mint a parazsat. Talán többen fel is tették maguknak a kérdést, vajon miféle veszélyt rejtenek. Nos, vessünk véget ennek a bizonytalanságnak. Nyomjuk meg valamelyik f billentyűt, és nézzük, mi történik!

Semmi.

Nyomhatjuk bármelyiket, akár együtt a SHIFTtel vagy a RUN/STOPpal, a CTRL vagy a C= segédbillentyűvel, akár egyedül, a világon semmi változást nem látunk a képernyőn. Ha egy meglevő programunkat betöltjük, és a futása közben nyomkodjuk az f gombokat, akkor sincs hatásuk. Ezek a billentyűk egyszerűen nem befolyásolják a számítógép működését - hacsak mi magunk nem adunk nekik szerepet, hatáskört, latin eredetű szóval: funkciót (ezért is áll rajtuk f betű: a funkció szó rövidítéseként).

A négy f billentyű tulajdonképpen nyolc: ezért is vannak 1-től 8-ig számozva. Önmagukban megnyomva f1, f3, f5, f7, a SHIFTtel együtt pedig f2, f4, f6, f8 a nevük.

A programjainkban azonban nem így kell hivatkoznunk rájuk, hanem a CHR$ számuk alapján. Íme egy kis táblázatban a nyolc f billentyű személyi adatai:

Billentyű	CHR$
f1	(133)
f3	(134)
f5	(135)
f7	(136)
f2	(137)
f4	(138)
f6	(139)
f8	(140)

Ha fellapozzuk a könyv végén a CHR$ táblázatot, meglátjuk, hogy az e számokhoz tartozó funkció helye üres; arra vár, hogy mi töltsük meg tartalommal. Mindjárt meg is látjuk, hogyan. Írjuk be ezt a kis programot:

```
0 REM FUNKCIOBILLENTYUK
10 GET A$:IF A$="" THEN 10
20 IF ASC(A$)<133 OR ASC(A$)>136 THEN 10
30 IF ASC(A$)=133 THEN GOSUB 100
40 IF ASC(A$)=134 THEN GOSUB 200
50 IF ASC(A$)=135 THEN GOSUB 300
60 IF ASC(A$)=136 THEN END
70 GOTO 10
100 POKE 53280,1
110 RETURN
200 POKE 53280,4
210 RETURN
300 POKE 53280,0
310 RETURN
```

Ebben a programban bőven el vagyunk látva újdonsággal. A 10-es sor mindjárt két érdekességet is kínál; az első a GET. Ez a szó angolul mindenfélét jelent, többek között azt is, hogy megkapni vagy megszerezni. Itt épp ebben a jelentésében szerepel.

A GET A$ tehát annyit tesz: szerezd meg az A$-t. Az A$ pedig, mivel értéke a programban nincs meghatározva, a billentyűzetről érkezik. Ennyiben hasonlít a már jól ismert INPUThoz, de egy különbseg van köztük: a GET nem várja meg, amíg megnyomjuk a RETURN billentyűt, hanem azonnal továbblépteti a programot. Ebből következik, hogy a GET utasitassal

csak egyetlen karakterre (billentyűre, CHR$-re, ahogy tetszik) utalhatunk. Ebből nem következik ugyan, de ide tartozik még egy fontos tudnivaló: a GET akkor sem vár, ha nem nyomunk meg semmilyen billentyűt. Ahhoz, hogy várja meg a billentyűnyomást, külön utasítást kell adni neki. Hogy ezt hogyan kell, már benne van programunk 10-es sorában:

```
10 GET A$:IF A$="" THEN 10
```



A két, szorosan egymás mellett álló idézőjel valóban az, aminek látszik: a semmit jelenti; egy tartalom nélküli stringet. Tehát esetünkben azt, hogy az A$-nek nincs értéke, mert nem nyomtunk meg semmilyen gombot.

Erre az esetre az IF...THEN szerkezet azt az utasítást adja a számítógépnek, hogy menjen a 10-es sorra. (Itt eláraultunk még egy egyszerűsítési lehetőséget is: a THEN után a Commodore BASICben elmaradhat a GOTO, a gép így is megérti az utasítást.) A program engedelmesen megy is a 10-es sor elejére - vagyis oda, ahol azt találja, hogy GET A$, satöbbi. Így jár a 10-es sorban körbe-körbe mindaddig, amíg meg nem nyomunk egy billentyűt.

Ha pedig megnyomunk egyet, akkor a gép elkezdi kiértékelni a beadott adatot. A 20-as sor arra szolgál, hogy az összes többi billentyűt letiltsa, csak a négy funkcióbillentyűt hagyja érvényesülni. (Nyilvánvaló: ha a megnyomott billentyű ASC értéke kisebb, mint az f1 gombé, vagy magasabb, mint az f7-é, akkor a programot azonnal visszaküldjük a 10-es sorra, magyarán: más

billentyűt nem veszünk figyelembe, csak azt a bizonyos négyet.)

Ezután következnek a GOSUBok. Az f1 billentyű kódja a 100-as sorban kezdődő szubrutinra küldi a programot, az f3 a 200-asra, az f5 a 300-asra, az f7-től pedig a program befejeződik.

A szubrutinokból a RETURN utasítás hatására a program mindig visszatér oda, ahol a GOSUB utasítást kapta, onnan az IF-es kifejezések "nem" ágán keresztül a 70-es sorra, amely visszaküldi a 10-esre, hogy várja a további gombnyomásokat - lásd mint fent.

Hogy ez a program mit csinál, azt egyszerűen nem vagyunk hajlandók elárulni - a szubrutinokban benne van, csak ki kell olvasni, és visszaemlékezni arra, hogy az első kötet második napján miről volt szó.

De ha még bírjuk, és képesek vagyunk egy új utasítást megtanulni, a fenti programot jelentősen leegyszerűsíthetjük az ON utasítás használatával.

```
0 REM FUNKCIOBILLENTYUK MASKEPP
10 GET A$:IF A$="" THEN 10
20 IF ASC(A$)=136 THEN END
30 K%=ASC(A$)-132
40 ON K% GOSUB 100,200,300
50 GOTO 10
100 POKE 53280,1
110 RETURN
200 POKE 53280,4
210 RETURN
300 POKE 53280,0
310 RETURN
```

Látjuk: egyetlen sorban három GOSUB utasítást adtunk ki az ON segítségével.

E szó jelentését az angolban sokkal könnyebb megadni, mint a BASICben. Angolul ugyanazt jelenti ugyanis, mint magyarul: -on. (On the monitor = a monitoron.) Ez esetben, a BASICben azonban egy rugalmas működésű szó: azt jelenti, hogy a hozzá kapcsolt változó (minálunk a K%) értékének megfelelően hajtsa végre a következő utasítást. Ha tehát K%=1, akkor a GOSUB után első helyen álló számú sorra (azaz a 100-asra) ugorjon, ha K%=2, akkor a 200-asra, ha pedig K%=3, akkor a 300-asra.

Az ON csak egyesével tud számolni, egytől felfelé, ezért a K% értékét a 30-as sorban úgy határoztuk meg, hogy a megnyomott funkcióbillentyűk ASC kódjának számértékét lecsökkentettük 132-vel. (Tudjuk: ezek közül a legkisebb 133, és egyesével növekszik.)

Ezt az utasítást nemcsak a GOSUBbal, hanem a GOTOval is használhatjuk, ha egy változó értékétől függően egy helyről különböző helyekre akarjuk ugratni programunkat.

Végezetül próbáljuk meg összegyúrni a legutóbbi két programunkat! Rakja az új program az általunk megadott sorrendbe a mondat szavait, emellett legyen lehetőségünk arra, hogy megváltoztassuk a keret (f1) és a háttér (f3) színét, az f5 megnyomására új számsorrendet kérjen, az f7 pedig állítsa le a program futását.

```
0 REM MIND ERTELMES?
10 POKE 53280,5:POKE 53281,13
20 PRINT"■"
30 A$(1)="EN"
40 A$(2)="EZT"
50 A$(3)="MONDTAM"
60 A$(4)="MAR"
70 A$(5)="EGYSZER"
80 A$(6)="TALAN"
90 A$(7)="NEKED"
100 GOSUB 160
110 GET C$:IF C$="" THEN 110
120 IF ASC(C$)=136 THEN END
130 K%=ASC(C$)-132
140 ON K% GOSUB 400,500,300
150 GOTO 110
160 REM KIHIVAS
170 PRINT"  TUDOK 7 OLYAN SZOT,"
180 PRINT"  AMELYEKBOL HA MONDATOKAT"
190 PRINT"  KESZITESZ, MINDEN MONDATOD"
200 PRINT"  ERTELMES LESZ."
210 PRINT"HA NEM HISZED, JARJ UTANA!"
220 RETURN
300 REM VALASZTOTT SZAMSOR
310 PRINT:PRINT
320 PRINT"IRD IDE A SZAMOKAT 1-TOL 7-IG"
330 PRINT"  TETSZOLEGES SORRENDBEN!"
```

```
340 FOR I=1 TO 7:INPUT B(I):NEXT I
350 PRINT:FOR I=1 TO 7
360 PRINT" "A$(B(I));
370 NEXT I:PRINT"."
380 RETURN
400 REM KERETSZINEZO
410 Z%=INT(16*RND(1))
420 POKE 53280,Z%
430 RETURN
500 REM HATTERSZINEZO
510 S%=INT(16*RND(1))
520 POKE 53281,S%
530 RETURN
```

Említettük, hogy némelyik változónév foglalt, nem használható. Azt azonban nem mondtuk meg, hogy mi célra foglaltak. Kettőről most mindjárt ki fog derülni.

```
10 TI$="000000"
20 PRINT TI$
30 PRINT "⌃⌃":REM CRSR FEL KETSZER
40 GOTO 20
```

Mielőtt elindítanánk ezt a a programot, áruljuk el, mi ez a TI$.

A számítógép belsejében egy óra működik, amely, ha másként nem rendelkezünk, kérésre a gép bekapcsolásától mért időt írja ki a képernyőre.

Próbáljuk ki ezt az órát! Gépeljük be:

```
? TI$
```

és RETURN. Most nem tudjuk pontosan megmondani, mit ír ki a gép, mert ez attól függ, hogy mióta van bekapcsolva. Mindenesetre egy hatjegyű számot látunk a képernyőn, amelyből az első kettő az órákat, a második kettő a perceket,a harmadik kettő a másodperceket jelenti.

Ennek az órának a mindenkori állását írathatjuk ki a képernyőre a TI és a TI$ változókkal. A TI$ a digitális karórák által is mutatott rendszer szerint, a TI (ez, a nevéből is kiderül, valós változó) hatvanad másodpercenként. Próbáljuk ki ezt is!

```
? TI
```

és RETURN. Ekkor egy négy-öt-hatjegyű számot ír ki a gép; ez TI pillanatnyi értéke. Ha felmegyünk a cursorral a ? TI sorra, és ismét megnyomjk a RETURNt, a

számot tartalmazó sorban új szám íródik ki a régi helyére, és ez annyival lesz magasabb, ahány hatvanad másodperc eltelt közben.

A gépnek ez a tulajdonsága remek stopperóra-(program megírására ad lehetőséget. Kezdjük az alappal.

Ha bekapcsoljuk a gépet, betöltjük a stopperórás programot, és elindítjuk, ezalatt jó néhány másodperc eltelik. Márpedig a stopperórát nulláról kell indítani, különben nincs értelme.

Erre természetesen gondoltak a gép konstruktőrei is. Első programsorunk legyen a TI$ értékének nullára állítása. Ez igen egyszerű:

```
10 TI$="000000"
```

Most még semmi nem történt, hiszen csak egy programsort írtunk; az óra által mutatott idő csak akkor áll majd nullára, ha a programot elindítjuk.

Most írassuk ki a TI mindenkori értékét. Tudjuk, hogy a TI hatvanadmásodpercekben számol, a TI$ pedig órában, percben, másodpercben. Az utóbbi közelebb áll hozzánk, ezért használjuk azt.

```
20 ? TI$
```

Itt annak az utasításnak kell következnie, amelyik a cursort visszaküldi az előző képernyősorra, hogy az óra egyhelyben álljon a képernyőn, ne egymás alá íródjanak a különböző értékek. Ehhez vissza kell léptetni a cursort.

```
30 ?" CRSR FEL KETSZER "
```

És végül vissza kell fordulni a program elejére, TI$ következő értékeért:

```
40 GOTO 20
```

Készen is vagyunk. A programot elindíthatjuk, a stopper a RUN szó alatti sorban működésbe lép, és "000000"-tól szépen számol fölfelé. Ha elérte a "000059"-es értéket, nem "000060" következik, hanem "000100", tehát a percekhez hozzáad egyet, a másodperceket nulláról újra kezdi.

Egy valamirevaló stopperóra persze legalább tizedmásodperceket mér; amihez már a TI$ nem alkalmas, hiszen az másodpercenként lép tovább. Elő kell vennünk a TI változót, és meg kell szelídítenünk.

Ha a TI hatvanadmásodperceket mér, akkor ebből másodperceket úgy kapunk, ha elosztjuk a mindenkori értékét hatvannal. Írjuk át programunk 20-as sorát így:

```
20 PRINT TI/60
```

Az így elindított program egy irgalmatlanul hosszú számot ír ki, vagy kilenc tizedesjeggyel, és a tizedesponttól balra egyesével számol felfelé. A tizedesjegyekből csak az elsőre van szükségünk, a többit elhagyhatjuk, hiszen úgysem tudjuk szemmel követni őket, meg aztán úgyis csak hármasok meg hatosok vannak benne (hiszen hat többszörösével osztottunk). Hagyjuk hát el őket. Emlékszünk, az RND függvénnyel kapcsolatban találkoztunk az INT kifejezéssel, amely törtszámokból leválasztotta a tizedesjegyeket. Használjuk fel most is ezt.

```
20 PRINT INT(TI/60)
```

Még mindig nem jutottunk tovább, mint ahonnan elindultunk: most a program körülbelül úgy működik, ahogyan a TI$ változóval működő programunk működött, azzal a hátránnyal, hogy a 60-as érték után 61-et ír ki, tehát csak másodpercben számol, percben nem. Ráadásul még mindig nem látjuk sehol a tizedmásodperceket.

No, a tizedmásodpercek gondját most már könnyen megoldjuk. Egy tizedesjegyet ki kell mentenünk az INT hatása alól. Ezt úgy csináljuk, hogy a TI értékét két részletben osztjuk hatvannal: előbb hattal, aztán tízzel, de az INT a tízzel való osztásra nem vonatkozik, ezt az egy tizedesjegyet engedélyezni fogja. Ez könnyebb, mint hinnénk: csak egy zárójelet kell ügyesen beiktatnunk:

```
20 PRINT (INT(TI/6))/10
```

A program még mindig csak másodpercben számol, a percekről nem vesz tudomást, de a tizedeket már kiírja.

Egy perc alatti időket már tudunk mérni vele.

Ez persze még mindig csak része a sikernek. Most azt kell elérnünk, hogy a gép kiírja a percek értékét, ehhez minden hatvanadik másodpercben hozzáadjon egyet, a másodperceket pedig állítsa nullára.

Hogy áttekinthetőbb legyen a változok rendszere, a másodperceket nevezzük el MP-nek, a perceket P-nek!

Írjuk újra a programot így:

```
0 PRINT CHR$(147):CLR:PRINT" 0 PERC";
10 TI$="000000"
20 MP=(INT(TI/6))/10
30 REM IDE MAJD IRUNK MEG VALAMIT
40 PRINT TAB(12)MP," MASODPERC"
50 PRINT "[T]"
60 GOTO 20
```

A nullás sorban látható új utasítás, a CLR (az angol **CLEAR**, ejtsd: klír; magyarul: "tiszta" szó rövidítése) **az összes** változó értékét nullára állítja - ezt a biztonság kedvéért tettük oda, mert már jó néhányszor átállítgattuk TI$ értékét, és bevezetünk új változókat is. Jobb, ha amikor futtatni kezdjük a programot, a változóink értéke mind 0.

Az így lefuttatott program üres kepernyővel kezd. Kiírja: 0 PERC, utána számlálgatja az egész és tizedmásodperceket, ezektől jobbra pedig még azt is kiírja: MASODPERC.

Hogyan oldható meg, hogy minden hatvanadik másodpercben a PERC szó előtt álló számjegy eggyel emelkedjék? Sejthető, hogy az IF...THEN szerkezetet kell használnunk, és ez igaz is. Írjunk be egy új 30-as sort!

```
30 IF MP=60 THEN 100
```

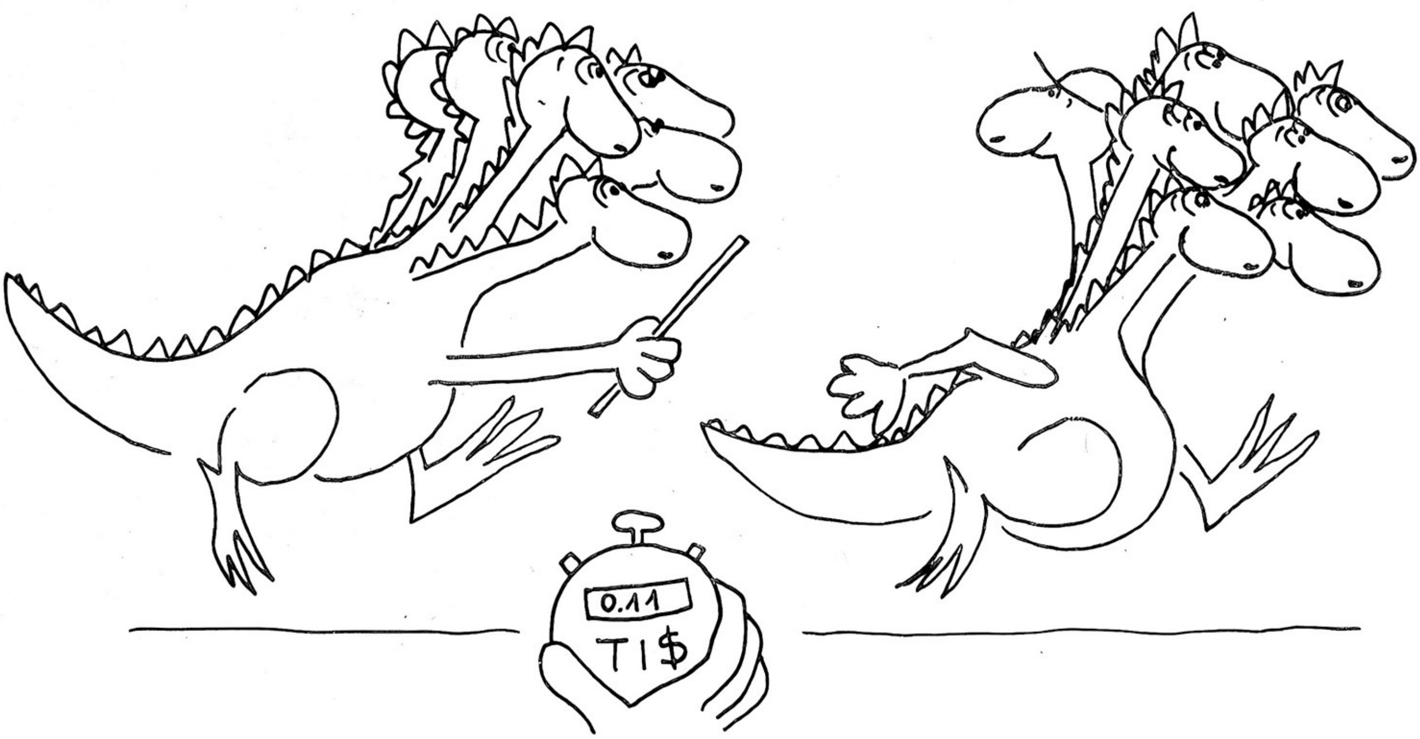
és egészítsük ki a programot két új sorral a végen:

```
100 P=P+1
200 PRINT P;: GOTO 10
```

Hát itt mit csináltunk? A P=P+1 egy eddig nem használt

változó értékét emeli meg eggyel. A CLR, mint mondtuk, mindent nullára állít, tehát ha új változót veszünk fel, az is 0-ról kezd, ha előtte más értéket nem adunk neki. Így amikor a programban az MP értéke először éri el a 60-at, és a GOTO leküldi a a programot a 100-as sorra, ehhez a 0-hoz ad hozzá 1-et a gép, és a képernyő bal sarkában álló 0 PERC felirat első karakterét kicseréli 1-re. Utána visszaugratja a programot arra a sorra, ahol a TI$ értékét nulláztuk le, ennélfogva a másodpercek 60-nál nem nőnek tovább, hanem újra indulnak 0-ról.

Kész a stopperóránk. Kivihetjük kazettára vagy lemezre, és csinálhatunk belőle olyan szubrutint is, amely egy program futása közben a játékos reakcióidejét méri. Egy ilyen felhasználását látjuk majd a hetedik nap programjai között.



Utolsó munkanapunkon végigkövettük tehát, hogyan születik meg egy meghatározott célú BASIC program, milyen gondolatmenettel építjük fel, hogy azt a célt szolgálja, amit szántunk neki. Utolsó gyakorlatként írjuk majd át úgy, hogy ha a percek száma eléri a 60-at, akkor órákat is számoljon! Az eddigiek ismeretében ez nem lesz nehéz.

A hátralévő egy nap arra szolgál, hogy néhány mintaprogram segítségével átismételjük, amit eddig megismertünk a Commodore 64 mikroszámítógép képességeiből.

A gép természetesen sokkal többet tud, mint amennyit egy ilyen 21 napos kis tanfolyamon meg lehet tanulni, de mostantól ha újságban, könyvben C 64-re írt BASIC programlistát látunk, nagy részét meg fogjuk érteni, és ez az első lépés ahhoz, hogy mind nagyobb mértékben tudjuk a saját szolgálatunkba állítani a gépet.

Az utolsó napon, ami e könyvből hátravan, vegyünk még néhány programot. Az első egy továbbfejlesztett változata a korábban már megismert csillagos programnak. A csillagok ezúttal már sokszínűek; a képernyő olykor tüzijátékhoz hasonlít.

Ez a program nem az összes színt használja, ezért akinek kedve van, átalakíthatja úgy, hogy a Commodore 64-nek mind a 16 színét tartalmazza!

```
0 REM SZINES CSILLAGOK
10 DIM A(15)
20 POKE 53280,0:POKE 53281,0
30 PRINT "♥":REM SHIFT+CLR/HOME
40 A(0)=5:A(1)=28:A(2)=30:A(3)=31
50 A(4)=129:A(5)=144
60 FOR I=6 TO 13:A(I)=I+143:NEXT I
70 A(14)=158:A(15)=159
80 P=INT(39*RND(1))
90 Q=INT(22*RND(1))
100 S=INT(15*RND(1))
110 FOR Y=0 TO Q:PRINT:NEXT Y
120 PRINT TAB(P)CHR$(A(S))"*"
130 FOR N=1 TO (10*RND(1)):NEXT N
140 PRINT"S"
150 GOTO 40
```

A második program egy rövid ellenőrzés, mennyire ismerjük a billentyűzetet, a karakterek elhelyezkedését. Ha a programlistát megfigyeljük, kiderül, hogy a játék itt időre megy!

```
0 REM BILLENTYUZET-TESZT
5 PRINT"♥"
10 PRINT"MENNYI IDO ALATT TALALOD MEG"
```

```
20 PRINT"A KARAKTEREKET?"
30 PRINT"HA MEGJELENIK EGY JEL, A LEHETO"
40 PRINT"LEGGYORSABBAN CSALD ELO MEG EGYSZER!"
50 PRINT"HA MEGUNTAD, NYOMD MEG AZ F1-ET!"
60 PRINT"NOSZA..."
70 KAR=INT(62*RND(0)+33)
80 PRINTTAB(5)CHR$(KAR),
90 TI$="000000"
100 GET A$:IF A$="" THEN GOTO 100
110 IF ASC(A$)=133 THEN END
120 IF A$<>CHR$(KAR) THEN GOTO 100
130 M%=VAL(TI$)
140 IF M%<60 THEN PRINTM%" MASODPERC":GOTO 70
150 P%=M%/100:MA%=M%-P%*100
160 PRINTP%" PERC"MA%" MASODPERC"
170 GOTO 70
```

Az itt következő programmal ábece-sorrendbe rendezhetünk bármilyen névsort, az utána következő másik változat pedig a betűrendbe szedett neveket ki is nyomtatja.

```
0 REM ABECE-SORREND
10 INPUT "A NEVEK SZAMA ";M
20 DIM A$(M+1)
30 FOR I=0 TO M-1:INPUT A$(I):NEXT I
40 FOR K=0 TO M-1
50 FOR L=K+1 TO M
60 IF A$(L)>=A$(K) THEN 100
70 S$=A$(L)
80 A$(L)=A$(K)
90 A$(K)=S$
100 NEXT L
110 NEXT K
120 PRINT:PRINT"BETURENDBEN:":PRINT
130 FOR I=0 TO M:PRINT A$(I):NEXT I
```

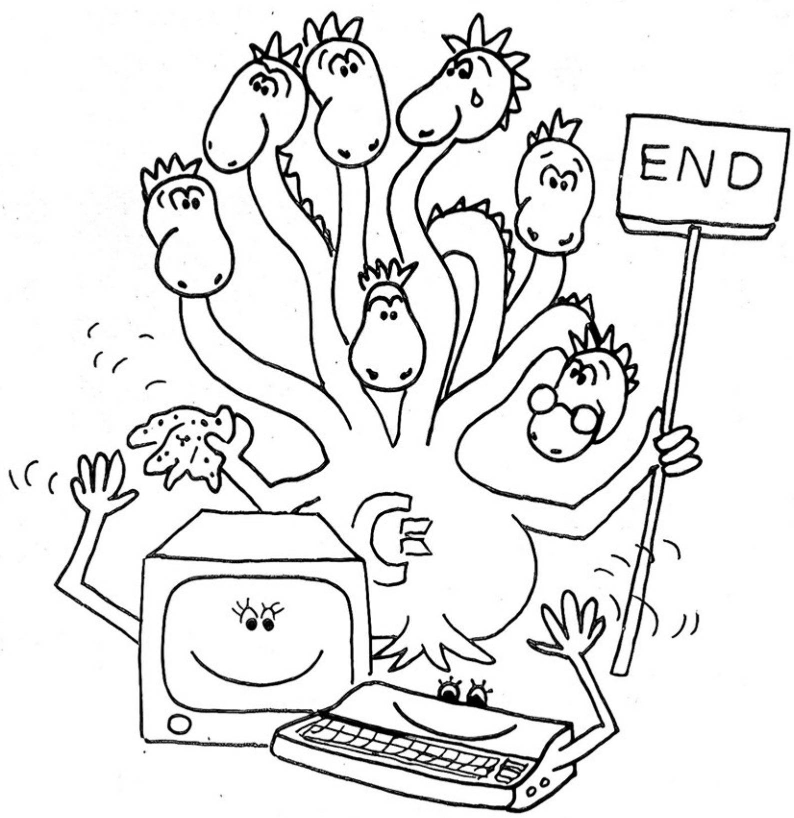
```
0 REM ABECE-SORREND PRINTERRE
5 PRINT"□"
10 INPUT "A SZAVAK SZAMA ";M
20 DIM A$(M+1)
30 FOR I=0 TO M-1:INPUT A$(I):NEXT I
40 FOR K=0 TO M-1
```

```
50 FOR L=K+1 TO M
60 IF A$(L)>=A$(K) THEN 100
70 S$=A$(L)
80 A$(L)=A$(K)
90 A$(K)=S$
100 NEXT L
110 NEXT K
115 OPEN1,4:CMD1
120 PRINT:PRINT "BETURENDBEN:":PRINT
130 FOR I=0 TO M:PRINT A$(I):NEXT I
140 PRINT#1:CLOSE1
```

Harmadik programunkban mi adhatunk fel rejtvényt a számítógépnek. Gondolunk egy számot 1 és 10 között, és a gép több kérdés után kitalálja. Ez a program eléggé bonyolult és hosszú, vigyázzunk a begépelésénél! Ha figyelmesen végigolvassuk, majdnem mindent megtalálunk benne, amiről eddig szó esett. Ezért utolsó ismétlő feladatnak is megfelel.

```
0 REM 1 ES 10 KOZOTT
5 PRINT"♥"
10 FOR I=1 TO 10:A(I)=I:NEXT I
20 PRINT:PRINT
30 PRINT"GONDOLJ EGY SZAMRA 1 ES 10 KOZOTT!"
40 INPUT"PRIMSZAM? (I/N)";K$
50 IF ASC(K$)=73 THEN GOTO 70
60 A(1)=0:A(2)=0:A(3)=0:A(5)=0:A(7)=0:GOTO 80
70 A(4)=0:A(6)=0:A(8)=0:A(9)=0:A(10)=0
80 INPUT"5-NEL NAGYOBB? (I/N)";K$
90 IF ASC(K$)=73 THEN 110
100 FOR I=6 TO 10:A(I)=0:NEXT I:GOTO 120
110 FOR I=1 TO 5:A(I)=0:NEXT I
120 GOSUB 600
130 IF K=1 THEN GOTO 10
140 INPUT"PAROS SZAM? (I/N)";K$
150 IF ASC(K$)=73 THEN GOTO 170
160 FOR I=0 TO 10 STEP 2:A(I)=0:NEXT I:GOTO 180
170 FOR I=1 TO 9 STEP 2:A(I)=0:NEXT I
180 GOSUB 600
190 IF K=1 THEN GOTO 10
191 INPUT"3-MAL OSZTHATO? (I/N)";K$
192 IF ASC(K$)=73 THEN GOTO 194
193 A(3)=0:A(6)=0:A(9)=0:GOTO 196
```

```
194 A(1)=0:A(2)=0:A(4)=0:A(5)=0:A(7)=0
195 A(8)=0:A(10)=0
196 GOSUB 600
197 IF K=1 THEN GOTO 10
200 INPUT"5-TEL OSZTHATO? (I/N)";K$
210 IF ASC(K$)=73 THEN GOTO 230
220 A(5)=0:A(10)=0:GOTO 250
230 FOR I=1 TO 4:A(I)=0:NEXT I
240 FOR I=6 TO 9:A(I)=0:NEXT I
250 GOSUB 600
260 GOTO 10
600 K=0
610 FOR I=1 TO 10:IF A(I)=0 THEN GOTO 630
620 K=K+1:B=A(I)
630 NEXT I
640 IF K<>1 THEN 700
650 PRINT"A KERESETT SZAM ="B
700 RETURN
```



TÁBLÁZATOK

SZÍNKEZELÉS

BILLENTYŰ	SZÍN	KÓD	KÉPERNYŐ-JEL
CTRL + 1	FEKETE	0	
CTRL + 2	FEHÉR	1	
CTRL + 3	PIROS	2	
CTRL + 4	CIÁNKÉK	3	
CTRL + 5	BÍBOR	4	
CTRL + 6	ZÖLD	5	
CTRL + 7	KÉK	6	
CTRL + 8	SÁRGA	7	
C= + 1	NARANCS	8	
C= + 2	BARNA	9	
C= + 3	RÓZSASZÍN	10	
C= + 4	SÖTÉTSZÜRKE	11	
C= + 5	KÖZÉPSZÜRKE	12	
C= + 6	VILÁGOSZÖLD	13	
C= + 7	VILÁGOSKÉK	14	
C= + 8	VILÁGOSSZÜRKE	15	

PRIORITÁS-TÁBLÁZAT

JEL	HASZNÁLAT	PÉLDA
↑	Hatványozás	ALAP↑Kitevo
-	Tagadás	-A
*/	Szorzás Osztás	AB*CD EF/GH
+-	Összeadás Kivonás	CNT+2 JK-PQ
>=<	Arányító műveletek	A< =B
NOT	Logikai NEM	NOT K%
AND	Logikai ÉS	JK AND 128
OR	Logikai VAGY	PQ OR 15

CURSORMOZGATÁS JELEI IDÉZŐJELEN BELÜL

BILLENTYŰ	JEL
CLR/HOME	
SHIFT + CLR/HOME	
CRSR LE	
CRSR FEL	
CRSR JOBBRA	
CRSR BALRA	

CHR$-TÁBLÁZAT

HATÁS	CHR$	HATÁS	CHR$	HATÁS	CHR$
	0	!	33	G	71
	1	"	34	H	72
	2	#	35	I	73
	3	$	36	J	74
	4	%	37	K	75
FEHÉR	5	&	38	L	76
	6	'	39	M	77
	7	(	40	N	78
SHIFT+C=		)	41	O	79
KIKAPCS.	8	*	42	P	80
SHIFT+C=		+	43	Q	81
BEKAPCS.	9	,	44	R	82
	10	-	45	S	83
	11	.	46	T	84
	12	/	47	U	85
RETURN	13	0	48	V	86
KKCS		1	49	W	87
F->A.	14	2	50	X	88
	15	3	51	Y	89
	16	4	52	Z	90
CRSR LE	17	5	53	[	91
RVS BE	18	6	54	£	92
CLR/		7	55	]	93
HOME	19	8	56	↑	94
INST/DEL	20	9	57	←	95
	21	:	58	—	96
	22	;	59	♠	97
	23	<	60	|	98
	24	=	61	—	99
	25	>	62	—	100
	26	?	63	—	101
	27	@	64	—	102
PIROS	28	A	65	|	103
CRSR		B	66	|	104
JOBB	29	C	67	╮	105
ZÖLD	30	D	68	╰	106
KÉK	31	E	69	╯	107
SZÓKÖZ	32	F	70	L	108

HATÁS	CHR$	HATÁS	CHR$	HATÁS	CHR$
╲	109	f4	138	▁	164
╱	110	f6	139	▏	165
┌	111	f8	140	▒	166
┐	112	SHIFT/		▕	167
●	113	RETURN	141	▒	168
_	114	KKCS		◤	169
♥	115	A->F	142	▕	170
│	116		143	├	171
╭	117	FEKETE	144	▗	172
╳	118	CRSR FEL	145	└	173
○	119	RVS KI	146	┐	174
♣	120	CLR/HOME	147	▂	175
│	121	INST/DEL	148	┌	176
◆	122	BARNA	149	┴	177
┼	123	RÓZSASZÍN	150	┬	178
▒	124	S. SZÜRKE	151	┤	179
│	125	K. SZÜRKE	152	▎	180
π	126	V. ZÖLD	153	▍	181
◥	127	V. KÉK	154	▐	182
	128	V. SZÜRKE	155	▔	183
NARANCS	129	BÍBOR	156	▀	184
	130	CRSR BAL	157	▃	185
	131	SÁRGA	158	┘	186
	132	CIÁNKÉK	159	▖	187
f1	133	NEGATÍV		▝	188
f3	134	SZÓKÖZ	160	┘	189
f5	135	▌	161	▘	190
f7	136	▄	162	▚	191
f2	137	▔	163		

CHR$ 192-223 ugyanaz, mint CHR$ 96-127
CHR$ 224-254 ugyanaz, mint CHR$ 160-190
CHR$ 255 ugyanaz, mint CHR$ 126

RÖVIDÍTÉSEK A TÁBLÁZATBAN:

KKCS = karakterkészlet-csere
F = felső
A = alsó
KIKAPCS. = kikapcsolás
BEKAPCS. = bekapcsolás

TARTALOM

ELŐKÉSZÜLETBEN

Hetedhét Sinclair Spectrum

66,– Ft